A faster algorithm for finding Tarski fixed points

John Fearnley
Department of Computer Science, University of Liverpool
john.fearnley@liverpool.ac.uk

Rahul Savani
Department of Computer Science, University of Liverpool
rahul.savani@liverpool.ac.uk

Abstract

Dang et al. have given an algorithm that can find a Tarski fixed point in a k-dimensional lattice of width n using $O(\log^k n)$ queries [2]. Multiple authors have conjectured that this algorithm is optimal [2, 7], and indeed this has been proven for two-dimensional instances [7]. We show that these conjectures are false in dimension three or higher by giving an $O(\log^2 n)$ query algorithm for the three-dimensional Tarski problem, which generalises to give an $O(\log^{k-1} n)$ query algorithm for the k-dimensional problem when $k \geq 3$.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases query complexity, Tarski fixed points, total function problem

1 Introduction

Tarski's fixed point theorem states that every order preserving function on a complete lattice has a greatest and least fixed point [11], and therefore in particular, every such function has at least one fixed point. Recently, there has been interest in the complexity of finding such a fixed point. This is due to its applications, including computing Nash equilibria of supermodular games and finding the solution of a simple stochastic game [7].

Prior work has focused on the complete lattice L defined by a k-dimensional grid of width n. Dang, Qi, and Ye [2] give an algorithm that finds a fixed point of a function $f : L \to L$ using $O(\log^k n)$ queries to f. This algorithm uses recursive binary search, where a k-dimensional problem is solved by making $\log n$ recursive calls on $(k-1)$-dimensional sub-instances. They conjectured that this algorithm is optimal.

Later work of Etessami, Papadimitriou, Rubinstein, and Yannakakis took the first step towards proving this [7]. They showed that finding a Tarski fixed point in a two-dimensional lattice requires $\Omega(\log^2 n)$ queries, meaning that the Dang et al. algorithm is indeed optimal in the two-dimensional case. Etessami et al. conjectured that the Dang et al. algorithm is optimal for constant k, and they leave as an explicit open problem the question of whether their lower bound can be extended to dimension three or beyond.

Our contribution. In this paper we show that, surprisingly, the Dang et al. algorithm is not optimal in dimension three, or any higher dimension, and so we falsify the prior conjectures. We do this by giving an algorithm that can find a Tarski fixed point in three dimensions using $O(\log^2 n)$ queries, thereby beating the $O(\log^3 n)$ query algorithm of Dang et al. Our new algorithm can be used as a new base case for the Dang et al. algorithm, and this leads to a $O(\log^{k-1} n)$ query algorithm for k-dimensional instances when $k \geq 3$, which saves a $\log n$ factor over the $O(\log^k n)$ queries used by the Dang et al. algorithm.

The Dang et al. algorithm solves a three-dimensional instance by making recursive calls to find a fixed point of $\log n$ distinct two-dimensional sub-instances. Our key innovation is to point out that one does not need to find a fixed point of the two-dimensional sub-instance to make progress. Instead, we define the concept of an *inner algorithm* (Definition 3) that, given a two-dimensional sub-instance, is permitted to return any point that lies in the up or down set of the three-dimensional instance (defined formally later). This is a much larger set of points, so whereas finding a fixed point of a two-dimensional instance requires $\Omega(\log^2 n)$ queries [7], we give a $O(\log n)$ query inner algorithm for two-dimensional instances. This inner algorithm is quite involved, and is the main technical contribution of the paper.

We show that, given an inner algorithm for dimension $k-1$, a reasonably straightforward *outer algorithm* can find a Tarski fixed point by making $O(k \cdot \log n)$ calls to the inner algorithm. Thus we obtain a $O(\log^2 n)$ query algorithm for the case where $k = 3$. We leave as an open problem the question of whether efficient inner algorithms exist in higher dimensions.

Though we state our results in terms of query complexity for the sake of simplicity, it should be pointed out that both our outer and inner algorithms run in polynomial time. Specifically, our algorithms will run in $O(\text{poly}(\log n, k) \cdot \log^{k-1} n)$ time when the function is presented as a Boolean circuit of size $\text{poly}(\log n, k)$.

Related work. Etessami et al. also studied the computational complexity of the Tarski problem [7], showing that the problem lies in PPAD and PLS. However, the exact complexity of the problem remains open. It is not clear whether the problem is PPAD $\cap$ PLS-complete [8], or contained in some other lower class such as EOPL or UEOPL [9].

Tarski's fixed point theorem has been applied in a wide range of settings within Economics [12, 10, 13], and in particular to settings that can be captured by supermodular games,

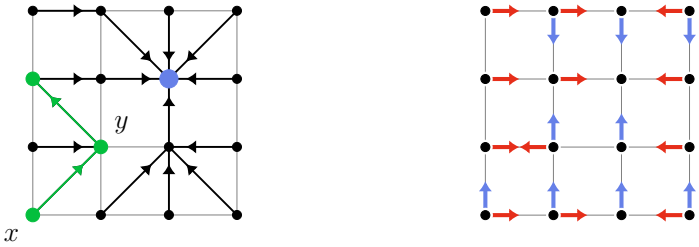

Figure 1 Left: a TARSKI instance. Right: our diagramming notation for the same instance.

which are in fact equivalent to the Tarski problem [7]. In terms of algorithms, Echenique [6] studied the problem of computing all pure equilibria of a supermodular game, which is at least as hard as finding the greatest or least fixed point of the Tarski problem, which is itself NP-hard [7]. There have also been several papers that study properties of Tarski fixed points, such as the complexity of deciding whether a fixed point is unique [2, 4, 3, 5]. The Tarski problem has also been studied in the setting where the partial order is given by an oracle [1].

2 Preliminaries

Lattices. We work with a complete lattice defined over a k-dimensional grid of points. We define $\text{Lat}(n_1, n_2, \ldots, n_k)$ to be the k-dimensional lattice with side-lengths given by $n_1, \ldots, n_k$. That is, $\text{Lat}(n_1, n_2, \ldots, n_k)$ contains every $x \in \mathbb{N}^k$ such that $1 \leq x_i \leq n_i$ for all $i = 1, \ldots, k$. Throughout, we use k to denote the dimensionality of the lattice, and $n = \max_{i=1}^{k} n_i$ to be the width of the widest dimension. We use $\preceq$ to denote the natural partial order over this lattice with $x \preceq y$ if and only if $x, y \in L$ and $x_i \leq y_i$ for all $i \leq k$.

The Tarski fixed point problem. Given a lattice L, a function $f : L \to L$ is *order preserving* if $f(x) \preceq f(y)$ whenever $x \preceq y$. A point $x \in L$ is *fixed point* of f if $f(x) = x$. A weak version of Tarski's theorem can be stated as follows.

▶ **Theorem 1** ([11]). *Every order preserving function on a complete lattice has a fixed point.*

Thus, we can define a total search problem for Tarski's fixed point theorem.

▶ **Definition 2** (TARSKI). *Given a lattice L, and a function $f : L \to L$, find one of:*

(T1) A point $x \in L$ such that $f(x) = x$.
(T2) Two points $x, y \in L$ such that $x \preceq y$ and $f(x) \not\preceq f(y)$.

Solutions of type (T1) are fixed points of f, whereas solutions of type (T2) witness that f is not an order preserving function. By Tarski's theorem, if a function f has no solutions of type (T2), then it must have a solution of type (T1), and so TARSKI is a total problem.

The left-hand picture in Figure 1 gives an example of a two-dimensional TARSKI instance. The blue point is a fixed point, and so is a (T1) solution, while the highlighted green arrows give an example of an order preservation violation, and so (x, y) is a (T2) solution.

Throughout the paper we will use a diagramming notation, shown on the right in Figure 1, that decomposes the dimensions of the instance. The red arrows correspond to dimension 1, where an arrow pointing to the left indicates that $f(x)_1 \leq x_1$, while an arrow to the right[1]

[1] If $x_1 = f(x)_1$ we could use either arrow, but will clarify in the text whenever this ambiguity matters.

indicates that $x_1 \leq f(x)_1$. Blue arrows do the same thing for dimension 2, and we will use green arrows for dimension 3 in the cases where this is relevant.

The up and down sets. Given a function f over a lattice L, we define $\text{Up}(f) = \{x \in L : x \preceq f(x)\}$, and $\text{Down}(f) = \{x \in L : f(x) \preceq x\}$. We call $\text{Up}(f)$, the *up set*, which contains all points in which f goes up according to the ordering $\preceq$, and likewise we call $\text{Down}(f)$ the *down set*. Note that the set of fixed points of f is exactly $\text{Up}(f) \cap \text{Down}(f)$.

Slices. A *slice* of the lattice L is defined by a tuple $s = (s_1, s_2, \ldots, s_k)$, where each $s_i \in N \cup \{*\}$. The idea is that, if $s_i \neq *$, then we fix dimension i of L to be s_i, and if $s_i = *$, then we allow dimension i of L to be free. Formally, we define the sliced lattice $L_s = \{x \in L : x_i = s_i \text{ whenever } s_i \neq *\}$. We say that a slice is a *principle slice* if it fixes exactly one dimension and leaves the others free. For example $(1, *, *)$, $(*, 33, *)$, and $(*, *, 261)$ are all principle slices of a three-dimensional lattice.

Given a slice s, and a function $f : L \to L$, we define $f_s : L_s \to L_s$ to be the *restriction* of f to L_s. Specifically, for each $x \in L_s$, we define $(f_s(x))_i = f(x)_i$ if $s_i = *$, and $(f_s(x))_i = s_i$ otherwise. This definition projects the function f down onto the slice s.

A fact that we will use repeatedly in the paper is that an order preservation violation in a slice s is also an order preservation violation for the whole instance. More formally, if $x, y \in L_s$ satisfy $x \preceq y$ and $f_s(x) \not\preceq f_s(y)$, then we also have $f(x) \not\preceq f(y)$, since there exists a dimension i such that $f(x)_i = f_s(x)_i > f_s(y)_i = f(y)_i$.

Sub-instances. A *sub-instance* of a lattice L is defined by two points $x, y \in L$ that satisfy $x \preceq y$. Informally, the sub-instance defined by x and y is the lattice containing all points between x and y. Formally, we define $L_{x,y} = \{a \in L : x \preceq a \preceq y\}$.

3 The Outer Algorithm

The task of the outer algorithm is to find a solution to the Tarski instance by making $O(k \cdot \log n)$ calls to the inner algorithm. We state our results for dimension k, even though we only apply the outer algorithm with $k = 3$, since, in the future, an efficient inner algorithm in higher dimensions may be found. Formally, an inner algorithm is defined as follows.

▶ **Definition 3** (Inner algorithm). *An inner algorithm for a Tarski instance takes as input a sub-instance $L_{a,b}$ with $a \in \text{Up}(f)$ and $b \in \text{Down}(f)$, and a principle slice s of that sub-instance. It outputs one of the following.*

- *A point $x \in L_{a,b} \cap L_s$ such that $a \in \text{Up}(f)$ or $b \in \text{Down}(f)$.*
- *Two points $x, y \in L_{a,b}$ that witness a violation of the order preservation of f.*

It is important to understand that here we are looking for points that lie in the up or down set of the *three-dimensional* instance, a point that lies in $\text{Up}(f_s)$ but for which f goes down in the third dimension would not satisfy this criterion.

The algorithm. Throughout the outer algorithm, we will maintain two points $x, y \in L$ with the invariant that $x \preceq y$ and $x \in \text{Up}(f)$ and $y \in \text{Down}(f)$. The following lemma, which is proved in Appendix A, implies that if x and y satisfy the invariant, then $L_{x,y}$ must contain a solution to the Tarski problem. This will allow us to focus on smaller and smaller instances that are guaranteed to contain a solution.

▶ **Lemma 4.** *Let L be a lattice and $f : L \to L$ be a Tarski instance. If there are two points $a, b \in L$ satisfying $a \preceq b$, $a \in \text{Up}(f)$, and $b \in \text{Down}(f)$, then one of the following exists.*

- *A point $x \in L_{a,b}$ satisfying $f(x) = x$.*

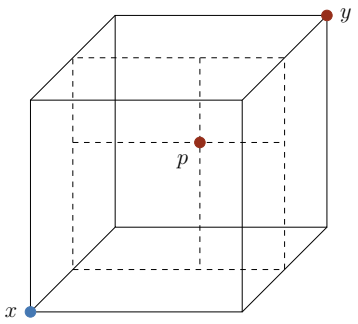

Figure 2 One iteration of the outer algorithm. The dashed lines show the principle slice chosen by the algorithm, and the point p is the point returned by the inner algorithm. In this case $p \in \text{Down}(f)$, and so the algorithm focuses on the sub-instance $L_{x,p}$.

- *Two points $x, y \in L_{a,b}$ satisfying $x \preceq y$ and $f(x) \not\preceq f(y)$.*

Moreover, there is an algorithm that finds one of the above using $O(\sum_{i=1}^{k}(a_i - b_i))$ queries.

Initially we set $x = (1, 1, \ldots, 1)$, which is the least element, and $y = (n_1, n_2, \ldots, n_k)$, which is the greatest element. Note that $x \preceq f(x)$ holds because x is the least element, and likewise $f(y) \preceq y$ holds because y is the greatest element, so the invariant holds for these two points.

Each iteration of the outer algorithm will reduce the number of points in $L_{x,y}$ by a factor of two. To do this, the algorithm selects a largest dimension of that sub-instance, which is a dimension i that maximizes $y_i - x_i$. It then makes a call to the inner algorithm for the principle slice s defined so that $s_i = \lfloor (y_i - x_i)/2 \rfloor$ and $s_j = *$ for all $j \neq i$.

1. If the inner algorithm returns a violation of order preservation in the slice, then this is also an order preservation violation in L, and so the algorithm returns this and terminates.
2. If the inner algorithm returns a point p in the slice such that $p \in \text{Up}(f)$, then the algorithm sets $x := p$ and moves to the next iteration.
3. If the inner algorithm returns a point p such that $p \in \text{Down}(f)$, then the algorithm sets $y := p$ and moves to the next iteration.

Figure 2 gives an example of this procedure.

The algorithm can continue as long as there exists a dimension i such that $y_i - x_i \geq 2$, since this ensures that there will exist a principle slice strictly between x and y in dimension i that cuts the sub-instance in half. Note that there can be at most $k \cdot \log n$ iterations of the algorithm before we arrive at the final sub-instance $L_{x,y}$ with $y_i - x_i < 2$ for all i. Lemma 4 gives us an efficient algorithm to find a solution in this final instance, which uses at most $O(\sum_{i=1}^{k}(y_i - x_i)) = O(k)$ queries. So we have proved the following theorem.

▶ **Theorem 5.** *Suppose that there exists an inner algorithm that makes at most q queries. Then a solution to the* Tarski *problem can be found by making $O(q \cdot k \cdot \log n + k)$ queries.*

4 The Inner Algorithm

We now describe an inner algorithm for three dimensions that makes $O(\log n)$ queries. Throughout this section we assume that the inner algorithm has been invoked on a sub-instance $L_{u,d}$ and principle slice s, and without loss of generality we assume that $s = (*, *, s_3)$.

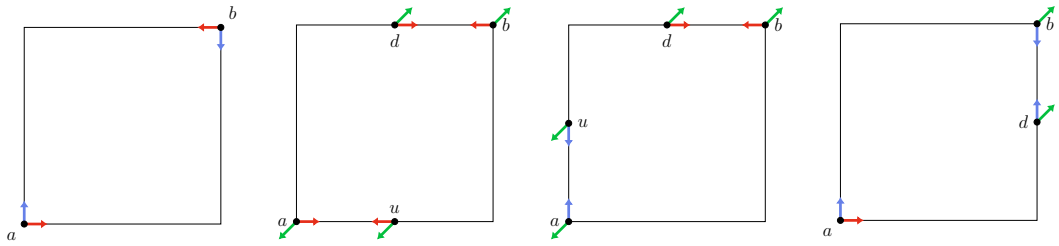

Figure 3 Four example sub-instances that satisfy the inner algorithm invariant.

Down set witnesses. Like the outer algorithm, the inner algorithm will also focus on smaller and smaller sub-instances that are guaranteed to contain a solution by an invariant, but now the invariant is more complex. To define the invariant, we first introduce the concept of a *down set witness* and an *up set witness*. The points d and b in the second example in Figure 3 give an example of a down set witness. Note that the following properties are satisfied.

- f weakly increases at d and b in dimension 3.
- d and b have the same coordinate in dimension 2.
- d weakly increases in dimension 1 while b weakly decreases in dimension 1.

We also allow down set witnesses like those given by d and b in the fourth example of Figure 3 that satisfy the same properties with dimensions 1 and 2 swapped. Thus, the formal definition of a down set witness abstracts over dimensions 1 and 2.

▶ **Definition 6** (Down set witness). *A down set witness is a pair of points (d, b) satisfying*

- $d_3 \leq f(d)_3$ *and* $b_3 \leq f(b)_3$.
- $\exists\ i, j \in \{1, 2\}$ *with* $i \neq j$ *s.t.* $d_i = b_i$ *and* $d_j \leq b_j$, *while* $d_j \leq f(d)_j$ *and* $f(b)_j \leq b_j$.

If (d, b) is a down set witness and $d_2 = b_2$, then we call (d, b) a *top-boundary* witness, while if $d_1 = b_1$, then we call (d, b) a *right-boundary* witness.

The following lemma states that if we have a down set witness (d, b), then between d and b we can find either a solution that can be returned by the inner algorithm (cases 1 and 2 of the lemma), or a point that is in the down set of the slice s (case 3 of the lemma).

The proof the lemma can be found in Appendix B. Informally, the proof for a top-boundary witness (d, b) uses the fact that d and b point towards each other in dimension 1 to argue that there must be a fixed point p (or an order preservation violation) of the one-dimensional slice between d and b. Then, it is shown that either this point is in $\mathrm{Up}(f)$, and so is a solution for the inner algorithm, or it is in $\mathrm{Down}(f_s)$, or that p violates order preservation with d or b.

▶ **Lemma 7.** *If (d, b) is a down set witness, then one of the following exists.*

1. *A point c satisfying $d \preceq c \preceq b$ such that $c \in \mathrm{Up}(f)$.*
2. *Two points x, y satisfying $d \preceq x \preceq y \preceq b$ that witness order preservation violation of f.*
3. *A point c satisfying $d \preceq c \preceq b$ such that $c \in \mathrm{Down}(f_s)$.*

Up set witnesses. An up set witness is simply a down set witness in which all inequalities have been flipped. The second and third diagrams in Figure 3 show the two possible configurations of an up set witness (a, u). Note that for up set witnesses, dimension 3 is now required to weakly decrease.

▶ **Definition 8** (Up set witness). *An up set witness is a pair of points (a, u) with $a, u \in L_s$ such that both of the following are satisfied.*

- $a_3 \geq f(a)_3$ *and* $u_3 \geq f(u)_3$.
- $\exists\ i, j \in \{1, 2\}$ *with* $i \neq j$ *s.t.* $a_i = u_i$ *and* $u_j \geq a_j$, *while* $u_j \geq f(u)_j$ *and* $f(a)_j \geq a_j$.

We say that an up set witness (a, b) is a *left-boundary* witness if $a_1 = b_1$, while we call it a *bottom-boundary* witness if $a_2 = b_2$.

The following lemma is the analogue of Lemma 7 for up set witnesses. The proof, which can be found in Appendix C, simply flips all inequalities in the proof of Lemma 7.

▶ **Lemma 9.** *If (a, u) is an up set witness, then one of the following exists.*

1. *A point c satisfying $a \preceq c \preceq u$ such that $c \in \text{Down}(f)$.*
2. *Two points x, y satisfying $a \preceq x \preceq y \preceq u$ that witness order preservation violation of f.*
3. *A point c satisfying $a \preceq c \preceq u$ such that $c \in \text{Up}(f_s)$.*

The invariant. At each step of the inner algorithm, we will have a sub-instance $L_{a,b}$ that satisfies the following invariant.

▶ **Definition 10** (Inner algorithm invariant). *The instance $L_{a,b}$ satisfies the invariant if*

- *Either $a \in \text{Up}(f_s)$ or there is a known up set witness (a, u) with $u \preceq b$.*
- *Either $b \in \text{Down}(f_s)$ or there is a known down set witness (d, b) with $a \preceq d$.*

If we have both an up set witness and a down set witness then we also require that $u \preceq d$.

Figure 3 gives four example instances that satisfy the invariant. Note that there are actually nine possible configurations, since the first point of the invariant can be satisfied either by a point in the up set, a left-boundary up set witness, or a bottom-boundary up set witness, and the second point of the invariant likewise has three possible configurations.

The following lemma shows that, if the invariant is satisfied, then the sub-instance $L_{a,b}$ contains a solution that can be returned by the inner algorithm. The proof invokes Lemmas 7 and 9 to either immediately find a solution for the inner algorithm, or find two points $x \preceq y$ in the sub instance where x is in the up set and y is in the down set. The latter case allows us to invoke Lemma 4 to argue that the sub-instance contains a fixed point p of the slice s. If p weakly increases in the third dimension, then $p \in \text{Up}(f)$, while if p decreases in the third dimension then $p \in \text{Down}(f)$. The full proof can be found in Appendix D.

▶ **Lemma 11.** *If $L_{a,b}$ satisfies the invariant then one of the following exists.*

- *A point $x \in L_{a,b}$ such that $x \in \text{Up}(f)$ or $x \in \text{Down}(f)$.*
- *Two points $x, y \in L_{a,b}$ that witness a violation of the order preservation of f.*

A special case. There is a special case that we will encounter in the inner algorithm that requires more effort to deal with. One example of this case is shown in Figure 4. Here we have a point p on the right-hand boundary of the instance that satisfies $p_1 < f(p)_1$, meaning that f moves p outside of the instance. If $b \in \text{Down}(f)$, or if there is a top-boundary down set witness, then it is straightforward to show that p and b violate order preservation.

However, if we have a right-boundary down set witness (d, b) with $p \preceq d$ then we need to do further work[2]. It can be shown that between the points d and b, there exists a point x

[2] The case where $p \succ d$ will never occur in our algorithm, so we can ignore it.

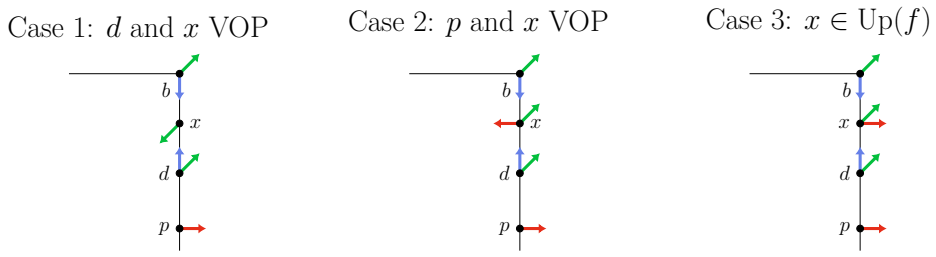

Figure 4 Example cases for Lemma 12. In the labels, VOP is short for "violate order preservation".

that is a fixed point of the one-dimensional slice, meaning that $x_2 = f(x)_2$. We show that this point can be found in $O(\log n)$ queries by applying binary search. Then, there are three cases, each of which is shown in Figure 4.

1. If f moves x strictly down in dimension 3, then d and x violate order preservation, since d is required to move upwards in dimension 3 by the properties of a down set witness.
2. If f moves x up in dimension 3 but strictly down in dimension 1, then p and x violate order preservation, since $p \preceq x$, but this implies that $f(p) \not\preceq f(x)$.
3. If f moves x up in dimensions 1 and 3, then $x \in \mathrm{Up}(f)$, since f does not move x in dimension 1.

In each case we obtain a solution that can be returned by the inner algorithm, though this does require us to spend $O(\log n)$ queries in order to find the point x.

The following lemma formally proves this, and also considers the three symmetric cases for when we have a top-boundary down set witness, or a left- or bottom-boundary up set witness. The proof can be found in Appendix E.

▶ **Lemma 12.** *Let $L_{a,b}$ be a sub-instance that satisfies the invariant, and let p be a point satisfying $a \preceq p \preceq b$ that also satisfies one of the following conditions.*

1. *$p_2 = b_2$, $p_2 < f(p_2)$, and there is a top-boundary down set witness (d, b) with $p \preceq d$.*
2. *$p_1 = b_1$, $p_1 < f(p_1)$, and there is a right-boundary down set witness (d, b) with $p \preceq d$.*
3. *$p_2 = a_2$, $p_2 > f(p_2)$, and there is a bottom-boundary up set witness (a, u) with $u \preceq p$.*
4. *$p_1 = a_1$, $p_1 > f(p_1)$, and there is a left-boundary up set witness (a, u) with $u \preceq p$.*

Then there exists a solution for the inner algorithm that can be found using $O(\log n)$ queries.

Initialization. The input to the algorithm is a sub-instance $L_{x,y}$, and recall that we have fixed the principle slice $s = (*, *, s_3)$. The initial values for a and b are determined as follows. For each dimension i we have $a_i = s_3$ if $i = 3$, and $a_i = x_i$ otherwise, and we have $b_i = s_3$ if $i = 3$, and $b_i = y_i$ otherwise. That is, a and b are the projections of x and y onto s.

The following lemma, whose proof can be found in Appendix F, states that either a and b satisfy the invariant, or that we can easily find a violation of order preservation.

▶ **Lemma 13.** *Either $L_{a,b}$ satisfies the invariant, or there is violation of order preservation between a and x, or between b and y.*

The algorithm. Now suppose that we have an instance $L_{a,b}$ that satisfies the invariant. We will describe how to execute one iteration of the algorithm, which will either find a violation of order preservation, or find a new instance whose size is at most half the size of the $L_{a,b}$.

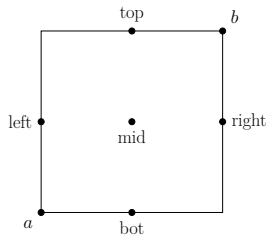

Figure 5 The five points used by the inner algorithm.

We begin by defining some important points. We define $\text{mid} = \lfloor (a+b)/2 \rfloor$ to be the *midpoint* of the instance, and we define the following points, which are shown in Figure 5:

$$\text{bot} = (\lfloor (a_1+b_1)/2 \rfloor, a_2), \qquad \text{left} = (a_1, \lfloor (a_2+b_2)/2 \rfloor),$$
$$\text{top} = (\lfloor (a_1+b_1)/2 \rfloor, b_2), \qquad \text{right} = (b_1, \lfloor (a_2+b_2)/2 \rfloor).$$

Step 1: Fixing the up and down set witnesses. Suppose that $L_{a,b}$ satisfies the invariant with a top-boundary down set witness (d, b), We would like to ensure that $\text{top} \preceq d$, since otherwise if we cut the instance in half in a later step, the witness may no longer be within the sub-instance. For the same reason, we would like to ensure that (d, b) satisfies $\text{right} \preceq d$ for a right-boundary down set witness, that (a, u) satisfies $u \preceq \text{left}$ for a left-boundary up set witness, and that (a, u) satisfies $u \preceq \text{bot}$ for a bottom-boundary up set witness. By the end of Step 1 we will have either found a violation of order preservation, moved into the next iteration with a sub-instance of half the size, or all inequalities above will hold.

Step 1 consists of the following procedure. The procedure should be read alongside Figure 6, which gives a diagram for every case presented below.

1. If (d, b) is a top-boundary down set witness and $\text{top} \preceq d$ then there is no need to do anything. On the other hand, if $d \prec \text{top}$ we use the following procedure.
   a. We first check if $\text{top}_3 > f(\text{top})_3$. If this is the case, then since the invariant ensures that $d_3 \leq f(d)_3$ we have $f(d)_3 \geq d_3 = \text{top}_3 > f(\text{top})_3$ so $d \preceq \text{top}$ but $f(d) \not\preceq f(\text{top})$, and an order preservation violation has been found and the inner algorithm terminates.
   b. We next check the whether $\text{top}_1 > f(\text{top})_1$. In this case, we can use (d, top) as a down set witness for the sub-instance $L_{a,\text{top}}$, where we observe that top satisfies the requirements since $\text{top}_3 \leq f(\text{top})_3$ and $\text{top}_1 > f(\text{top})_1$. Hence, $L_{a,\text{top}}$ satisfies the invariant (if $L_{a,b}$ also has an up set witness (a, u) then note that $u \preceq d$ continues to hold), and so the algorithm moves into the next iteration with the sub-instance $L_{a,\text{top}}$.
   c. In this final case we have $\text{top}_3 \leq f(\text{top})_3$ and $\text{top}_1 \leq f(\text{top})_1$. Therefore (top, b) is a valid down set witness for $L_{a,b}$ (if $L_{a,b}$ also has an up set witness (a, u) then note that $u \preceq d \prec \text{top}$). So we can replace (d, b) with (top, b) and continue, noting that our down set witness now satisfies the required inequality.
2. If (d, b) is a right-boundary down set witness and $\text{right} \preceq d$ then there is no need to do anything. On the other hand, if $d \prec \text{right}$ then we use the same procedure as case 1, where dimensions 1 and 2 are exchanged and the point top is replaced by the point right.
3. If (a, u) is a bottom-boundary up set witness and $u \preceq \text{bot}$ then there is no need to do anything. On the other hand, if $\text{bot} \prec u$ then we use the following procedure, which is the same as the procedure from case 1, where all inequalities have been flipped.

   a. We first check if $\text{bot}_3 < f(\text{bot})_3$. If this is the case, then since the invariant ensures that $u_3 \geq f(u)_3$ we have $f(u)_3 \leq u_3 = \text{bot}_3 < f(\text{bot})_3$ so $u \succeq \text{bot}$ but $f(u) \not\succeq f(\text{bot})$, and an order preservation violation has been found and the inner algorithm terminates.
   b. We next check the whether $\text{bot}_1 < f(\text{bot})_1$. In this case, we can use (bot, u) as an up set witness for the sub-instance $L_{\text{bot},b}$, where we observe that bot satisfies the requirements since $\text{bot}_3 \geq f(\text{bot})_3$ and $\text{bot}_1 < f(\text{bot})_1$. Hence, $L_{\text{bot},b}$ satisfies the invariant (if $L_{a,b}$ also has a down set witness (d, b) then note that $u \preceq d$ continues to hold), and so the algorithm moves into the next iteration with the sub-instance $L_{\text{bot},b}$.
   c. In this final case we have $\text{bot}_3 \geq f(\text{bot})_3$ and $\text{bot}_1 \geq f(\text{bot})_1$. Therefore (a, bot) is a valid up set witness for $L_{a,b}$ (if $L_{a,b}$ also has a down set witness (d, b) then we note that $\text{bot} \preceq u \preceq d$). So we can replace (a, u) with (a, bot), noting that our up set witness now satisfies the required inequality.

4. If (a, u) is a left-boundary up set witness and $u \preceq \text{left}$ then there is no need to do anything. On the other hand, if $u \succ \text{left}$ then we use the same procedure as case 3, where dimensions 1 and 2 are exchanged and the point left is replaced by the point bot.

Step 2: Find a smaller sub-instance. If Step 1 of the algorithm did not already move us into the next iteration of the algorithm with a smaller instance, we apply Step 2. This step performs a case analysis on the point mid. The following procedure should be read in conjunction with Figure 7, which provides a diagram for every case.

1. Check if $\text{mid}_1 \leq f(\text{mid})_1$ and $\text{mid}_2 \leq f(\text{mid})_2$. If this is the case then $\text{mid} \in \text{Up}(f_s)$, and so we can move to the next iteration of the algorithm with the sub-instance $L_{\text{mid},b}$. Note that if $L_{a,b}$ has a down-set witness (d, p), then Step 1 of the algorithm has ensured that $\text{mid} \preceq d$, and so (d, p) is also a valid down-set witness for $L_{\text{mid},b}$.
2. Check if $\text{mid}_1 \geq f(\text{mid})_1$ and $\text{mid}_2 \geq f(\text{mid})_2$. If this is the case then $\text{mid} \in \text{Down}(f_s)$, and so we can move to the next iteration of the algorithm with the sub-instance $L_{a,\text{mid}}$. Note that if $L_{a,b}$ has an up-set witness (a, u), then Step 1 of the algorithm has ensured that $u \preceq \text{mid}$, and so (a, u) is also a valid down-set witness for $L_{a,\text{mid}}$.
3. Check if $\text{mid}_1 \leq f(\text{mid})_1$ and $\text{mid}_2 > f(\text{mid})_2$. If so, we use the following procedure.

   a. Check if $\text{mid}_3 \leq f(\text{mid})_3$. If so, do the following.
      i. Check if $\text{right}_3 > f(\text{right})_3$. If this holds then we have $f(\text{mid})_3 \geq \text{mid}_3 = \text{right}_3 > f(\text{right})_3$, meaning that $\text{mid} \preceq \text{right}$ but $f(\text{mid}) \not\preceq f(\text{right})$. Thus we have found a violation of order preservation and the algorithm terminates.
      ii. Check if $\text{right}_1 < f(\text{right})_1$. If this holds then we use Lemma 12 to find a solution that can be returned by the inner algorithm with $O(\log n)$ further queries.
      iii. If we reach this case then we have $\text{mid}_3 \leq f(\text{mid})_3$ and $\text{right}_3 \leq f(\text{right})_3$, while we also have $\text{mid}_1 \leq f(\text{mid})_1$ and $\text{right}_1 \geq f(\text{right})_1$. Thus $(\text{mid}, \text{right})$ is a valid down set witness for the instance $L_{a,\text{right}}$. Note that if $L_{a,b}$ has an up set witness (a, u), then Step 1 ensures that $u \preceq \text{mid}$, and so $L_{a,\text{right}}$ satisfies the invariant. So the algorithm moves to the next iteration with sub-instance $L_{a,\text{right}}$.
   b. In this case we have $\text{mid}_3 > f(\text{mid})_3$. The following three steps are symmetric to those used in Case 3.a, but with all inequalities flipped, dimension 1 substituted for dimension 2, the point bot substituted for right, and the point a substituted for b.
      i. Check if $\text{bot}_3 > f(\text{bot})_3$. If this holds then we have $f(\text{mid})_3 \leq \text{mid}_3 = \text{bot}_3 < f(\text{bot})_3$, meaning that $\text{mid} \succeq \text{bot}$ but $f(\text{mid}) \not\succeq f(\text{bot})$. Thus we have found a violation of order preservation and the algorithm terminates.

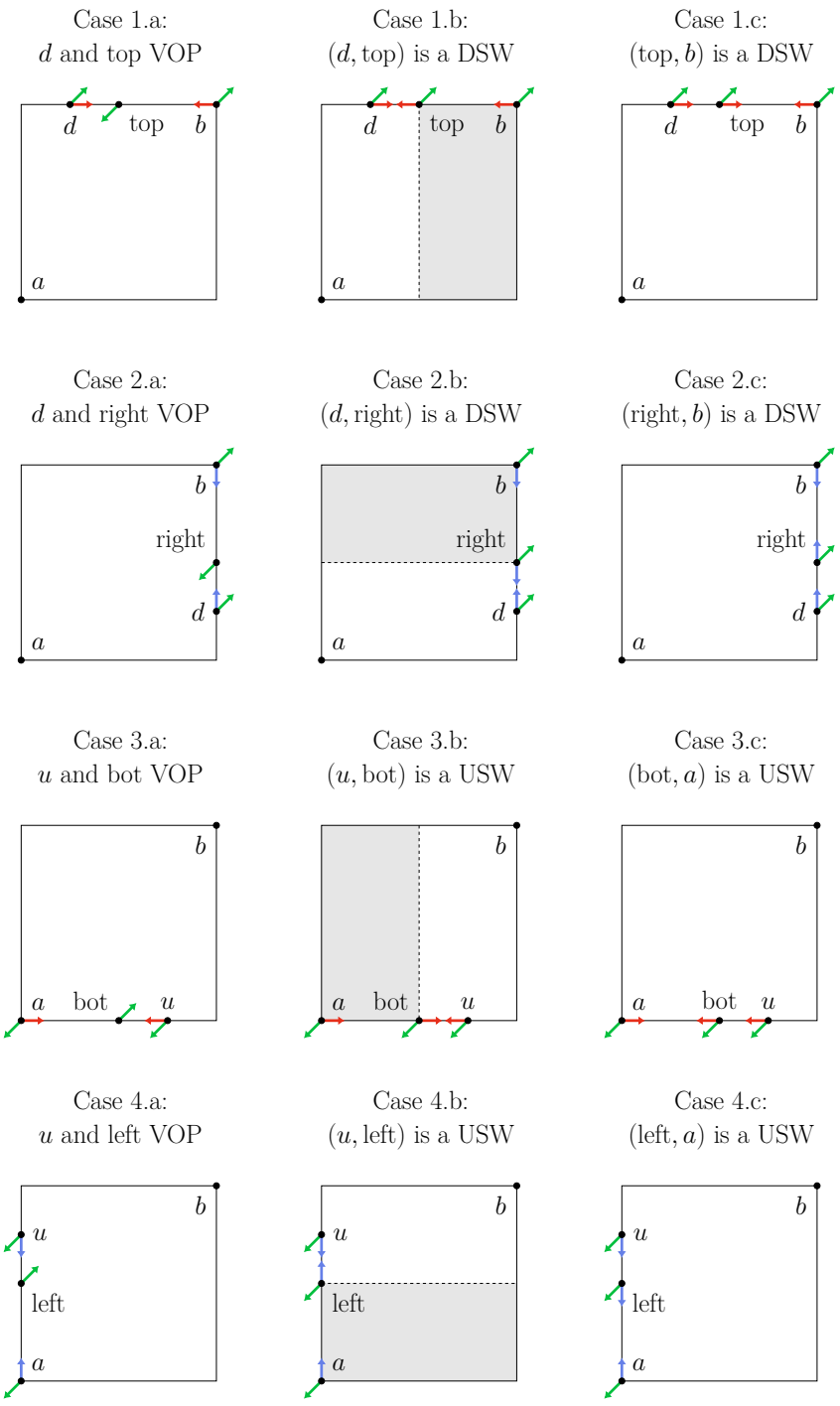

Figure 6 All cases used in Step 1 of the algorithm. In the labels, VOP is short for "violate order preservation", DSW is short for "down set witness", and USW is short for "up set witness".

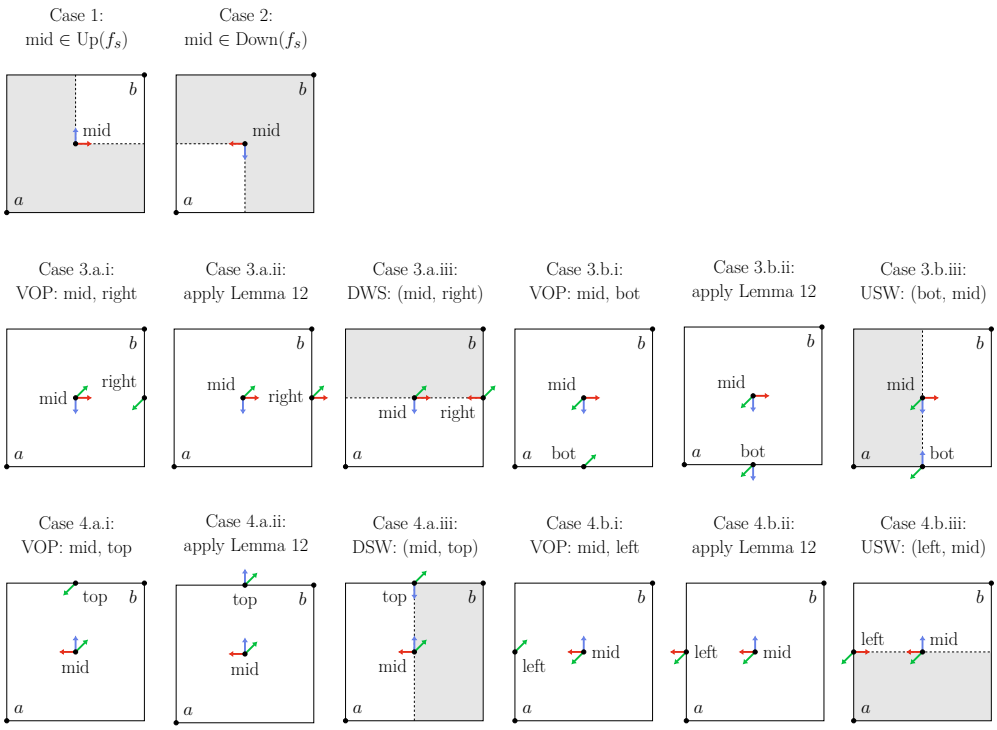

Figure 7 All cases used in Step 2 of the algorithm. In the labels, VOP is short for "violate order preservation", DSW is short for "down set witness", and USW is short for "up set witness".

ii. Check if $\text{bot}_2 > f(\text{bot})_2$. If this holds then we can use Lemma 12 to find a solution that can be returned by the inner algorithm by making $O(\log n)$ further queries.

iii. If we reach this case then we have $\text{mid}_3 \geq f(\text{mid})_3$ and $\text{bot}_3 \geq f(\text{bot})_3$, while we also have $\text{mid}_2 \geq f(\text{mid})_2$ and $\text{bot}_2 \leq f(\text{bot})_2$. Thus (bot, mid) is a valid up set witness for the instance $L_{\text{bot},b}$. Note that if $L_{a,b}$ has a down set witness (d, b), then Step 1 of the algorithm ensures that $d \succeq \text{mid}$, and so $L_{\text{bot},b}$ satisfies the invariant. The algorithm will therefore move to the next iteration with the sub-instance $L_{\text{bot},b}$.

4. In this final case we have $\text{mid}_1 > f(\text{mid})_1$ and $\text{mid}_2 \leq f(\text{mid})_2$. Here we follow the same procedure as Case 3, but with dimensions 1 and 2 exchanged, every instance of the point right replaced with top, and every instance of bot replaced with left.

It is possible that in some iteration we have $b_i \leq a_i + 1$ for some index i, and thus we may have right $= b$ or bot $= a$, or other similar equalities. We note that the algorithm will continue to work even if these equalities hold, although some of the checks will become redundant (eg. the check in Case 3.a.ii cannot succeed if right $= b$ due to the invariant).

So, if the algorithm does not hit any of the cases that return a solution immediately, then it can continue until it finds an instance $L_{a,b}$ with $b_1 \leq a_1 + 1$ and $b_2 \leq a_2 + 1$ that satisfies the invariant. Lemma 11 implies that any sub-instance that satisfies the invariant contains a solution that can be returned by the inner algorithm. Since then $L_{a,b}$ contains at most four points, we can check all of them and then return the solution that must exist.

Query complexity. Observe that each iteration of the algorithm either finds a violation of order preservation, applies Lemma 12 to find a solution using $O(\log n)$ further queries, or reduces the size of one of the dimensions by a factor of two. Moreover, each non-terminating

iteration of the algorithm queries exactly five points. Hence, if the algorithm is run on a sub-instance $L_{a,b}$ with $n_1 = b_1 - a_1$ and $n_2 = b_2 - a_2$, then the algorithm will terminate after making at most $O(\log n_1 + \log n_2 + \log n)$ queries. So the overall query complexity of the algorithm is $O(\log n)$, and we have shown the following theorem.

▶ **Theorem 14.** *There is an $O(\log n)$-query inner algorithm for 3-dimensional* Tarski.

Theorems 5 and 14 imply that 3-dimensional Tarski can be solved using $O(\log^2 n)$ queries, and this can be combined with the $\Omega(\log^2 n)$ lower bound for two-dimensional Tarski [7], to give the following theorem, where the straightforward lower bound is proved in Appendix G.

▶ **Theorem 15.** *The deterministic query complexity of three-dimensional* Tarski *is $\Theta(\log^2 n)$.*

5 Extension to higher dimensions

We now extend our results to show that k-dimensional Tarski can be solved using $O(\log^{k-1} n)$ queries. The algorithm of Dang et al. [2] solves a k-dimensional Tarski instance by making $O(\log n)$ recursive calls to an algorithm for solving $(k-1)$-dimensional Tarski instances. Our algorithm can be plugged into this recursion as a new base case for $k = 3$.

The following lemma is a consequence of the work of Dang et al. [2]. However, their algorithm deals with the promise version of Tarski in which it is assumed that the input function is order preserving. For this reason, we provide our own proof of the lemma in which we give a variation of the algorithm that either finds a fixed point or explicitly provides a violation of order preservation. The proof of the following lemma can be found in Appendix H.

▶ **Lemma 16.** *If $(k-1)$-dimensional* Tarski *can be solved using q queries, then k-dimensional* Tarski *can be solved using $(q+2) \cdot (\log n + 2)$ queries.*

The direct consequence of Lemma 16 and our $O(\log^2 n)$ query algorithm for three-dimensional Tarski is the following theorem.

▶ **Theorem 17.** *k-dimensional* Tarski *can be solved using $O(\log^{k-1} n)$ queries for $k \geq 3$,*

Time complexity. To obtain time complexity results, note that writing down a point in the lattice L already requires $k \cdot \log n$ time. We assume that f is implemented by a Boolean circuit of size that is polynomial in k and $\log n$. With this assumption, our time complexity result can be stated as follows.

▶ **Theorem 18.** *If f is presented as a Boolean circuit of size* $\text{poly}(\log n, k)$*, then for $k \geq 3$ there is an algorithm for* Tarski *that runs in time $O(\text{poly}(\log n, k) \cdot \log(n)^{k-1})$.*

6 Conclusion

Our $O(\log^{k-1} n)$ query algorithm for k-dimensional Tarski falsifies prior conjectures that the problem required $\Omega(\log^k n)$ queries [2, 7]. This, of course, raises the question of what is the query complexity of finding a Tarski fixed point? While our upper bound is tight in three dimensions, it seems less likely to be the correct answer in higher dimensions. Indeed, there seems to be a fairly wide range of possibilities. Is it possible to show a $\log^{\Omega(k)} n$ query lower bound for the problem? Or perhaps there exists a fixed parameter tractable algorithm that uses $O(f(k) \cdot \log^2 n)$ queries? Both of those would be consistent with the known upper and lower bounds, and so further research will be needed to close the gap.

References

1 Ching-Lueh Chang, Yuh-Dauh Lyuu, and Yen-Wu Ti. The complexity of Tarski's fixed point theorem. *Theor. Comput. Sci.*, 401(1-3):228–235, 2008.

2 Chuangyin Dang, Qi Qi, and Yinyu Ye. Computations and complexities of Tarski's fixed points and supermodular games. *CoRR*, abs/2005.09836, 2020. Stanford tech report version appeared in 2012.

3 Chuangyin Dang and Yinyu Ye. On the complexity of a class of discrete fixed point problems under the lexicographic ordering. Technical report, 2018. Technical Report, City University of Hong Kong, CY2018-3, 17 pages.

4 Chuangyin Dang and Yinyu Ye. On the complexity of an expanded tarski's fixed point problem under the componentwise ordering. *Theor. Comput. Sci.*, 732:26–45, 2018.

5 Chuangyin Dang and Yinyu Ye. Erratum/correction to "on the complexity of an expanded tarski's fixed point problem under the componentwise ordering" [theor. comput. sci. 732 (2018) 26-45]. *Theor. Comput. Sci.*, 817:80, 2020.

6 Federico Echenique. Finding all equilibria in games of strategic complements. *J. Econ. Theory*, 135(1):514–532, 2007.

7 Kousha Etessami, Christos H. Papadimitriou, Aviad Rubinstein, and Mihalis Yannakakis. Tarski's theorem, supermodular games, and the complexity of equilibria. In *Proc. of ITCS*, volume 151, pages 18:1–18:19, 2020.

8 John Fearnley, Paul W. Goldberg, Alexandros Hollender, and Rahul Savani. The complexity of gradient descent: CLS = PPAD ∩ PLS. Technical report, 2020. Preprint.

9 John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. Unique end of potential line. *J. Comput. Syst. Sci.*, 114:1–35, 2020.

10 Paul Milgrom and John Roberts. Rationalizability, learning, and equilibrium in games with strategic complementarities. *Econometrica*, 58(6):1255–1277, 1990.

11 Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.

12 Donald M. Topkis. Equilibrium points in nonzero-sum n-person submodular games. *SIAM J. Control Optim*, 17:773–787, 1979.

13 Donald M. Topkis. *Supermodularity and Complementarity*. Princeton University Press, 1998.

A Proof of Lemma 4

Before we prove Lemma 4, we first prove the following auxiliary lemma. It states that if x is in the up set, and i is a dimension for which f moves strictly upwards, then either the point x' that is directly above x in dimension i is also in the upset, or x and x' witness a violation of the order preservation of f.

▶ **Lemma 19.** *Let L be a lattice, $f : L \to L$ be a* Tarski *instance, $x \in L$ be a point satisfying $x \in \mathrm{Up}(f)$, and i be a dimension such that $x_i < f(x)_i$. Define the point x' so that*

$$x'_j = \begin{cases} x_j + 1 & \text{if } j = i, \\ x_j & \text{if } j \neq i. \end{cases}$$

Either $x' \in \mathrm{Up}(f)$, or x and x' witness a violation of the order preservation of f.

Proof. For dimension i, if $f(x')_i < x'_i$ then we have

$$f(x')_i < x'_i \leq f(x)_i,$$

where the second inequality follows from the assumption that $f(x)_i > x_i$, and the fact that $x'_i = x_i + 1$. Hence, if $f(x')_i < x'_i$, then x and x' witness a violation of the order preservation of f since $x \preceq x'$ but $x \not\preceq f(x')$.

For a dimension $j \neq i$, if $f(x')_j < x'_j$ then we have

$$f(x')_j < x'_j = x_j \leq f(x)_j,$$

where the equality follows from the fact that $x_j = x'_j$ by definition, and the final inequality follows from the assumption that $x \in \mathrm{Up}(f)$. Hence, if $f(x')_j < x'_j$, then we have that $x \preceq x'$ while $f(x) \not\preceq f(x')$, and so x and x' witness a violation of the order preservation of f.

From what we have proved above, we know that if x and x' do not violate order preservation, then we have $x'_j \leq f(x')_j$ for all dimensions j, which implies that $x' \in \mathrm{Up}(f)$. ◀

We now prove Lemma 4. Here we repeatedly apply Lemma 19 to generate a sequence of points in the up set that start at a and monotonically increase according to $\preceq$. If the path ends inside $L_{a,b}$ then we show that it must end at a solution, while if it leaves $L_{a,b}$, then we show that there is a violation of order preservation between b and the point at which the path leaves the sub-instance.

Proof. Since $a \in \mathrm{Up}(f)$, we have that either a is a fixed point of f, or there exists a dimension i such that $a_i < f(a)_i$. In the former case we are done, so let us assume that the latter is true. This means that we can apply Lemma 19 to obtain a point $a' = a + e_j$, where e_j is the unit vector in dimension j, such that either a and a' witness an order preservation violation of f, or $a' \in \mathrm{Up}(f)$.

By repeatedly applying the argument above, we can construct a sequence of points

$$a = a_1, a_2, a_3, \ldots, a_p$$

such that for all $i < p$ we have that and $a_i = a_{i-1} + e_j$ for some dimension j, and $a_i \in \mathrm{Up}(f)$. The sequence ends at the point a_p where we can no longer apply the argument, which means that either a_p and a_{p-1} witness the order preservation violation of f, or there is no index j such that $(a_p)_j < f(a_p)_j$, meaning that a_p is a fixed point since $a_p \in \mathrm{Up}(f)$. Note that the sequence cannot be infinite since $a_i \prec a_{i+1}$ for all i, and the lattice is finite.

If $a_p \preceq b$ then we are immediately done, since this implies that we have one of the required solutions in the sub-instance $L_{a,b}$. On the other hand, if $a_p \not\preceq b$, then let i be the largest index such that $a_i \preceq b$, and let j be the dimension such that $a_{i+1} = a_i + e_j$. Note that we have $b_j = (a_i)_j$, since otherwise we would have $a_{i+1} \preceq b$. We have

$$f(b)_j \leq b_j = (a_i)_j < f(a_i)_j,$$

where the first inequality holds because $b \in \mathrm{Down}(f)$, and the final inequality holds because $a_{i+1} = a_i + e_j$, which can only occur when $(a_i)_j < f(a_i)_j$. Hence we have $a_i \preceq b$, but $f(a_i) \not\preceq f(b)$, and so a_i and b witness a violation of the order preservation of f. Furthermore, both a_i and b lie in the sub-instance $L_{a,b}$, and so the proof is complete.

For the algorithm, note that the arguments above imply that it is sufficient to find either a_p, or the index i such that $a_i \preceq b$ and $a_{i+1} \not\preceq b$ (i must be unique if it exists, since $a_j \prec a_{j+1}$ for all j). Each element of the sequence can be constructed by making a single query to f, and since each step of the sequence strictly increases one coordinate of the point, we have that there can be at most $\sum_{i=1}^{k}(a_i - b_i)$ points a_i of the sequence satisfying $a_i \preceq b$. Thus the algorithm makes at most

$O(\sum_{i=1}^{k}(a_i - b_i))$ queries. ◀

B Proof of Lemma 7

Proof. Observe that, since $d_i = b_i$, we have that d and b both lie in the same one-dimensional slice. Moreover, the fact that $d_j \leq f(d)_j$ implies that d lies in the up set of this slice, while $f(b)_j \leq b_j$ implies that b lies in the down set of this slice. Hence, we can apply Lemma 4 to either find a fixed point p of this one-dimensional slice, or a violation of order preservation. In the latter case, we are done since case two of the lemma will have been fulfilled, so we will proceed assuming that p exists.

There are now two cases to consider.

- **Case 1:** $\mathbf{p_i \leq f(p)_i}$. There are two sub-cases.
  - **Sub-case 1:** $\mathbf{p_3 \leq f(p)_3}$. Note that $p_j = f(p)_j$, since p is the fixed point of the one-dimensional slice. Hence $p \in \mathrm{Up}(f)$, and so the first case of the lemma has been fulfilled.
  - **Sub-case 2:** $\mathbf{p_3 > f(p)_3}$. Now we have

    $$f(p)_3 < p_3 = d_3 \leq f(d)_3,$$

    where the equality holds because d and p both lie in s, and the final inequality holds by the requirements of a down set witness. Therefore we have $d \preceq p$ but $f(d) \not\preceq f(p)$, and so d and p witness a violation of the order preservation of f, and so the second case of the lemma is satisfied.
- **Case 2:** $p_i > f(p)_i$. Note that $p_j = f(p)_j$ since p is a fixed point of the one-dimensional slice. Hence p is in the down set of the slice s, and so the third condition of the lemma has been fulfilled.

◀

C Proof of Lemma 9

Proof. This proof is exactly the same as the proof of Lemma 7, but all inequalities have been flipped. We include it only for the sake of completeness.

Observe that, since $a_i = u_i$, we have that a and u both lie in the same one-dimensional slice. Moreover, the fact that $u_j \geq f(u)_j$ implies that u lies in the down set of this slice, while $f(a)_j \geq a_j$ implies that a lies in the up set of this slice. Hence, we can apply Lemma 4 to either find a fixed point p of this one-dimensional slice, or a violation of order preservation. In the latter case, we are done since case two of the lemma will have been fulfilled, so we will proceed assuming that p exists.

There are now two cases to consider.

- **Case 1:** $\mathbf{p_i \geq f(p)_i}$. There are two sub-cases.
  - **Sub-case 1:** $\mathbf{p_3 \geq f(p)_3}$. Note that $p_j = f(p)_j$, since p is the fixed point of the one-dimensional slice. Hence $p \in \mathrm{Down}(f)$, and so the first case of the lemma has been fulfilled.
  - **Sub-case 2:** $\mathbf{p_3 < f(p)_3}$. Now we have

    $$f(p)_3 > p_3 = a_3 \geq f(a)_3,$$

    where the equality holds because a and p both lie in s, and the final inequality holds by the requirements of an up set witness. Therefore we have $a \preceq p$ but $f(a) \not\preceq f(p)$, and so a and p witness a violation of the order preservation of f, and so the second case of the lemma is satisfied.

- **Case 2:** $\mathbf{p_i < f(p)_i}$. Note that $p_j = f(p)_j$ since p is a fixed point of the one-dimensional slice. Hence p is the up set of the slice s, and so the third condition of the lemma has been fulfilled.

◀

D Proof of Lemma 11

Proof. We can find a point $x \in L_{a,b}$ that satisfies $x \in \text{Up}(f_s)$ in the following way. By the invariant, either a already satisfies this condition, or we can apply Lemma 9 to obtain one of the three possible cases from that lemma. Cases one and three immediately fulfill the requirements of this lemma, and so we are done immediately in those cases, while the second case gives us the point x.

Symmetrically, we can find a point $y \in L_{a,b}$ that satisfies $y \in \text{Down}(f_s)$, since either b is such a point, or we can invoke Lemma 7 to either immediately fulfil the requirements of this lemma, or produce the point y.

Note further that we have $x \preceq y$. If either $a = x$ or $b = y$ then this holds due to the promises given by the invariant. When we have both an up and down set witness we have $x \preceq u \preceq d \preceq y$, where the first and third inequalities are come from Lemmas 9 and 7, while $u \preceq d$ is promised by the invariant.

Hence we can apply Lemma 4 to the sub-instance $L_{x,y} \subseteq L_{a,b}$, which will either give us a violation of order preservation, which will immediately satisfy the second condition of this lemma, or a fixed point $p \in L_{x,y}$ of the slice s. We now do a case analysis on the third dimension.

- If $p_3 \leq f(p)_3$, then $p \in \text{Up}(f)$ since $p_1 = f(p_1)$ and $p_2 = f(p_2)$.
- If $p_3 > f(p)_3$, then $p \in \text{Down}(f)$ since $p_1 = f(p_1)$ and $p_2 = f(p_2)$.

Hence, in either case the first condition of this lemma is satisfied. ◀

E Proof of Lemma 12

We first prove the following auxiliary lemma.

▶ **Lemma 20.** *All of the following are true.*

1. *Given a top-boundary down set witness (d, b) there is a $O(\log n)$ query algorithm to find a point x satisfying $d \preceq x \preceq b$ such that $x_1 = f(x)_1$ or a violation of order preservation.*
2. *Given a right-boundary down set witness (d, b) there is a $O(\log n)$ query algorithm to find a point x satisfying $d \preceq x \preceq b$ such that $x_2 = f(x)_2$ or a violation of order preservation.*
3. *Given a bottom-boundary up set witness (a, u) there is a $O(\log n)$ query algorithm to find a point x satisfying $a \preceq x \preceq u$ such that $x_1 = f(x)_1$ or a violation of order preservation.*
4. *Given a left-boundary up set witness (a, u) there is a $O(\log n)$ query algorithm to find a point x satisfying $a \preceq x \preceq u$ such that $x_2 = f(x)_2$ or a violation of order preservation.*

Proof. All four algorithms can be implemented by binary search. We give the algorithm explicitly for the first case, and we will then explain what needs to be changed for the other three cases.

The algorithm maintains two variables x^l and x^r, which will satisfy the invariant that $x^l_1 \leq f(x^l)_1$ and $x^r_1 \geq f(x^r)_1$. Initially we set $x^l = d$ and $x^r = b$. This satisfies the invariant since $d_1 \leq f(d)_1$ and $b_1 \geq f(b)_1$ are required to hold for all top-boundary down set witnesses.

In each step we construct the point $x^m = \lfloor (d+b)/2 \rfloor$. If $x_1^m \leq f(x^m)_1$ then we set $x^l = x^m$, and otherwise we $x^r = x^m$. Observe that this choice ensures that the invariant continues to hold. Therefore, after at most $\log n$ iterations we will have arrived at two adjacent points x^l, x^r that satisfy the invariant and also satisfy $x^r = x^l + 1$. If $x_1^l = f(x^l)_1$ or $x_1^r = f(x^r)_1$ then we are done. Otherwise, note that the inequalities in the invariant must be strict for both x^l and x^r, so we have

$$f(x^l)_1 > x_1^l = x_1^r - 1 \geq f(x^r)_1,$$

so we have $x^l \preceq x^r$ but $f(x^l) \not\preceq f(x^r)$ and therefore x^l and x^r violate order preservation.

For the other three claims we can use the same algorithm with the following changes.

- For the second claim we use the same algorithm but swap dimensions 1 and 2.
- For the third claim we use the same algorithm but substitute (a, u) for (d, b).
- For the fourth claim we use the same algorithm but substitute (a, u) for (d, b) and swap dimensions 1 and 2.

◀

Now we proceed to prove Lemma 12.

Proof. We consider four cases, which each correspond to one of the four possible preconditions of the lemma.

1. In the first case we suppose that we have $p_2 = b_2$ and $p_2 < f(p_2)$. We now consider the possible ways that the point b can satisfy the invariant.
   a. If $b \in \text{Down}(f_s)$ or if there is a right-boundary down set witness (d, b), then we know that $f(b)_2 \leq b_2$. This then implies

      $$f(p)_2 > p_2 = b_2 \geq f(b)_2,$$

      so we have $p \preceq b$ but $f(p) \not\preceq f(b)$, and so p and b violate order preservation.
   b. If there is a top-boundary down set witness (d, b) then we can apply Lemma 20 to obtain a point x satisfying $d \preceq x \preceq b$ such that $x_1 = f(x)_1$ in $O(\log n)$ queries.
      i. If $x_3 > f(x)_3$ then we have

         $$f(d)_3 \geq d_3 = x_3 > f(x)_3,$$

         where the first inequality is a requirement for (d, b) to be a down set witness. Hence $d \preceq x$ but $f(d) \not\preceq f(x)$ and so d and x violate order preservation.
      ii. If $x_3 \leq f(x)_3$ but $x_2 \geq f(x)_2$ then we have

         $$f(p)_2 > p_2 = d_2 = x_2 \geq f(x)_2,$$

         where the first inequality came from the precondition of the lemma. Hence $p \preceq d \preceq x$ but $f(p) \not\preceq f(x)$ and so p and x violate order preservation.
      iii. If $x_3 \leq f(x)_3$ and $x_2 < f(x)_2$ then $x \in \text{Up}(f)$ since $x_1 = f(x)_1$. Therefore x can be returned by the inner algorithm.
2. In the second case we have $p_1 = b_1$ and $p_1 < f(p_1)$. For this case we can use the same procedure as case 1, but with dimensions 1 and 2 swapped.
3. In the third case we have $p_2 = a_2$ and $p_2 > f(p_2)$. This case can follow the procedure from case 1, where all inequalities have been flipped.

a. If $a \in \text{Up}(f_s)$ or if there is a left-boundary up set witness (a, u), then we know that $f(a)_2 \geq a_2$. This then implies

$$f(p)_2 < p_2 = a_2 \leq f(a)_2,$$

so we have $p \succeq a$ but $f(p) \not\succeq f(a)$, and so p and a violate order preservation.

b. If there is a bottom-boundary down set witness (a, u) then we can apply Lemma 20 to obtain a point x satisfying $a \preceq x \preceq u$ such that $x_1 = f(x)_1$ in $O(\log n)$ queries.

i. If $x_3 < f(x)_3$ then we have

$$f(u)_3 \leq u_3 = x_3 < f(x)_3,$$

where the first inequality is a requirement for (a, u) to be an up set witness. Hence $u \succeq x$ but $f(u) \not\succeq f(x)$ and so u and x violate order preservation.

ii. If $x_3 \geq f(x)_3$ but $x_2 \leq f(x)_2$ then we have

$$f(p)_2 < p_2 = u_2 = x_2 \leq f(x)_2,$$

where the first inequality came from the precondition of the lemma. Hence $p \succeq u \succeq x$ but $f(p) \not\succeq f(x)$ and so p and x violate order preservation.

iii. If $x_3 \geq f(x)_3$ and $x_2 > f(x)_2$ then $x \in \text{Down}(f)$ since $x_1 = f(x)_1$. Therefore x can be returned by the inner algorithm.

4. In the final case we have $p_1 = a_1$ and $p_1 > f(p_1)$. Here we can apply the procedure from case 3 with dimensions 1 and 2 exchanged.

◀

F Proof of Lemma 13

Proof. We will show that either $a \in \text{Up}(f_s)$ and $b \in \text{Down}(f_s)$, or that a violation of order preservation can be found.

We start by showing that either we have a violation of order preservation, or we have that $a \in \text{Up}(f_s)$. To check this, we only need to inspect dimensions $i \in \{1, 2\}$. Note that if $f(a)_i < a_i$ for some i, then

$$f(a)_i < a_i = x_i \leq f(x)_i,$$

where the equality holds from the definition of a, and the final inequality holds since $x \in \text{Up}(f)$ is a requirement for calling the inner algorithm. Thus, if $f(a)_i < a_i$ for some dimension $i \in \{1, 2\}$ then we have $x \preceq a$ and $f(x) \not\preceq f(a)$, and so we have that x and a witness a violation of the order preservation of f. Otherwise, we have $f(a)_i \leq a_i$ for all $i \in \{1, 2\}$, and therefore $a \in \text{Up}(f_s)$.

We can apply the same reasoning symmetrically to b and y. If $b_i > f(b)_i$ for some $i \in \{1, 2\}$ then

$$f(b)_i > b_i = y_i \geq f(y)_i,$$

where the equality holds from the definition of b and the final inequality holds since $y \in \text{Down}(f)$ is a requirement for calling the inner algorithm. Thus we would have $b \preceq y$ and $f(b) \not\preceq f(y)$, and so either b and y witness a violation of the order preservation of f, or $b \in \text{Down}(f_s)$.

At this stage we have either satisfied the second or third conditions of the lemma, or we have that $a \in \text{Up}(f_s)$ and $b \in \text{Down}(f_s)$ and so the invariant on $L_{a,b}$ is satisfied. ◀

G Proof of Theorem 15

Proof. We will show that, if solving a $(k-1)$-dimensional Tarski instance requires q queries, then solving k-dimensional Tarski also requires q queries.

Let $L^{k-1} = \mathrm{Lat}(n_1, n_2, \ldots, n_{k-1})$ be a $(k-1)$-dimensional lattice, and let $f^{k-1} : L^{k-1} \to L^{k-1}$ be a Tarski instance over L^{k-1} that requires q queries to solve. Let $L^k = \mathrm{Lat}(n_1, n_2, \ldots, n_k)$ be a k-dimensional lattice, where n_k is any positive integer. We build the function $f^k : L^k \to L^k$ in the following way. For each point $x \in L^k$ we define

$$f^k(x)_i = \begin{cases} f^{k-1}(x)_i & \text{if } i < k, \\ x_k & \text{if } i = k. \end{cases}$$

Observe that $x = f^k(x)$ if and only if x is also a fixed point of f^{k-1}, and that x and y violate the order preservation of f^k if and only if x and y violate the order preservation of f^{k-1}. Moreover, every query to f^k can be answered by making exactly one query to f^{k-1}. Hence, in order solve the Tarski problem for f^k, we must make at least q queries.

Thus, Theorem 15 follows from the $\Omega(\log^2 n)$ query lower bound of Etessami et al. [7] for Tarski in dimension 2. ◀

H Proof of Lemma 16

Proof. The algorithm maintains a sub-instance $L_{x,y}$ defined by two points $x \preceq y$ that satisfy the following invariant.

- x is either the least element, or $x_1 < f(x)_1$ and $f(x)_i = x_i$ for all $i > 1$.
- y is either the greatest element, or $y_1 > f(y)_1$ and $f(y)_i = y_i$ for all $i > 1$.

Initially we set x to be the least element, and y to be the greatest element. These points trivially satisfy the invariant.

In each step the algorithm fixes the slice $s = (\lfloor x_1 + y_1 \rfloor, *, \ldots, *)$. It then considers the points

$$a_i = \begin{cases} s_1 & \text{if } i = 1, \\ x_i & \text{if } i > 1. \end{cases} \qquad b_i = \begin{cases} s_1 & \text{if } i = 1, \\ y_i & \text{if } i > 1. \end{cases}$$

These are simply the projections of x and y onto the slice s. The algorithm makes the following checks.

- Does there exist a dimension $i > 1$ such that $a_1 > f(a)_1$? If there is then we have

  $$f(a)_i < a_i = x_i \leq f(x_i),$$

  and therefore we have $x \preceq a$ but $f(x) \not\preceq f(a)$. So x and a violate order preservation, and the algorithm can terminate.
- Does there exist a dimension $i > 1$ such that $b_1 < f(b)_1$? If there is then we have

  $$f(b)_i > b_i = y_i \geq f(y_i),$$

  and therefore we have $y \succeq b$ but $f(y) \not\succeq f(b)$. So y and b violate order preservation, and the algorithm can terminate.

If the algorithm did not terminate, then we have that $a \in \text{Up}(f_s)$ and $b \in \text{Down}(f_s)$. Then, by Lemma 4, we know that there is a fixed point p in the instance $L_{x,y} \cap L_s$. and since this is a $(k-1)$-dimensional instance, the fixed point p can be found using q queries.

Note that since p is a fixed point of the slice, we have that $p_i = f(p)_i$ for all $i > 1$. Considering dimension 1 gives us three cases.

- If $p_1 = f(p)_1$ then p is a fixed point and the algorithm can terminate and return p.
- If $p_1 < f(p)_1$ then the instance $L_{p,y}$ satisfies the invariant, so the algorithm continues to the next step with $L_{p,y}$.
- If $p_1 > f(p)_1$ then the instance $L_{x,p}$ satisfies the invariant, so the algorithm continues to the next step with $L_{x,p}$.

So either the algorithm immediately terminates, or it proceeds to the next iteration with an instance that is at most half the size. Note that each iteration requires $q+2$ queries, where the extra two queries were to the points a and b.

The algorithm can continue until is reaches a sub-instance $L_{x,y}$ where $y_1 = x_1 + 1$, at which point it can no longer cut the instance in half along dimension 1. Note that this will occur after at most $\log n$ iterations, and so $(q+2) \cdot \log n$ queries will have been made.

At this point, the algorithm now enters the final phase where the following operations are carried out.

1. If x is the least element, then the algorithm finds a fixed point p of the instance $L_{x,y} \cap L_s$ where $s = (1, *, \ldots, *)$ using the same procedure as above. This takes an additional $q+1$ queries (there is no need to check for order preservation violations between x and a since $x = a$).
   Note that we cannot have $p_1 > f(p)_1$ since $p_1 = 1$. So, if $p_1 = f(p)_1$ then p is a fixed point and the algorithm terminates, and if $p_1 < f(p)_1$ then $L_{p,y}$ satisfies the invariant, and so the algorithm proceeds with this sub-instance.
2. If y is the greatest element, then the algorithm finds a fixed point p of the instance $L_{x,y} \cap L_s$ where $s = (n, *, \ldots, *)$ using the same procedure as above. This takes an additional $q+1$ queries (there is no need to check for order preservation violations between y and b since $y = b$).
   Note that we cannot have $p_1 < f(p)_1$ since $p_1 = n$. So, if $p_1 = f(p)_1$ then p is a fixed point and the algorithm terminates, and if $p_1 > f(p)_1$ then $L_{x,p}$ satisfies the invariant, and so the algorithm proceeds with this sub-instance.
3. At this stage we now know that x is not the least element, and y is not the greatest element and that the invariant is satisfied. We have

   $$f(y)_1 < y_1 = x_1 + 1 \leq f(x)_1,$$

   where the first and third inequalities come from the invariant. Therefore we have $x \preceq y$ but $f(x) \not\preceq f(y)$, so x and y violate order preservation.

Thus, in total the algorithm uses

$$(q+2) \cdot \log n + 2 \cdot (q+1) \leq (q+2) \cdot (\log n + 2)$$

queries. ◀